Artificial Neural Networks and Deep Architectures, DD2437

Short report on lab assignment 4
Restricted Boltzmann Machines, Deep Belief Networks, and Stacked Autoencoders

Nicolas Oulianitski Essipova

27th February 2020

1 Main objectives and scope of the assignment

- Analysis of the Restricted Boltzmann Machine, RBM, and training using the contrastive divergence algorithm.
- Analysis of the Deep Belief Network, DBN, using greedy layer-wise pre-training, with two and three RBMs.
- Analysis of the Stacked Autoencoder, SA, and how it relates to a DBN.

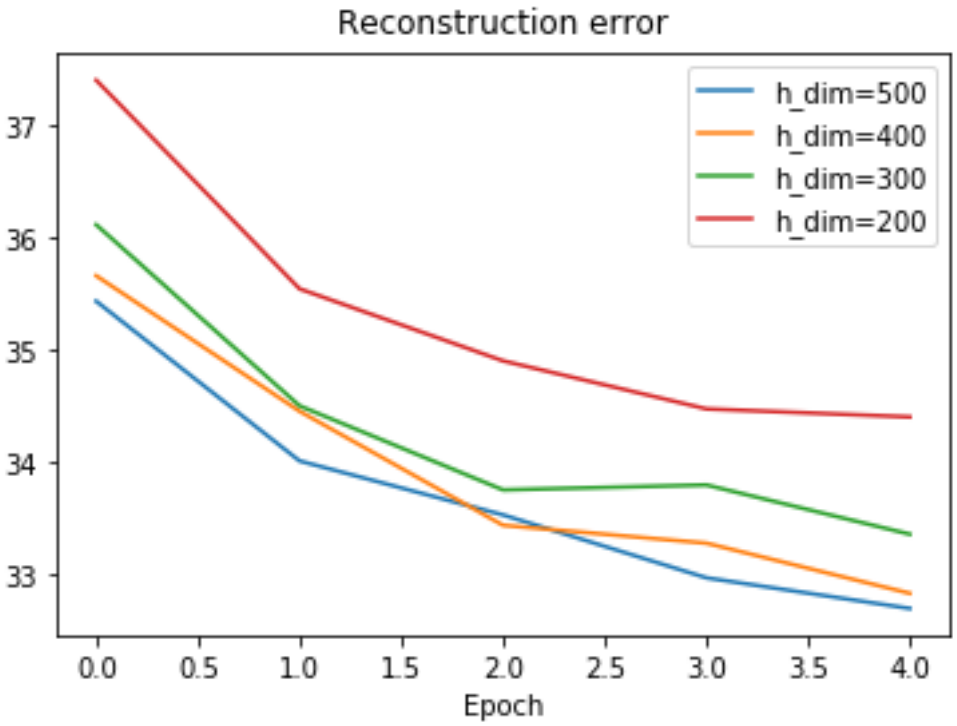
2 Methods

The lab was conducted within a Jupyter Notebook using Python 3.7 as the programming language, running on the Google Cloud Platform. Numpy was used for the numerical computations regarding implementation and analysis of RBM and DBN. PyTorch 1.0 was used for implementation and analysis of SA. Matplotlib was used for visualization.

3 Results and discussion

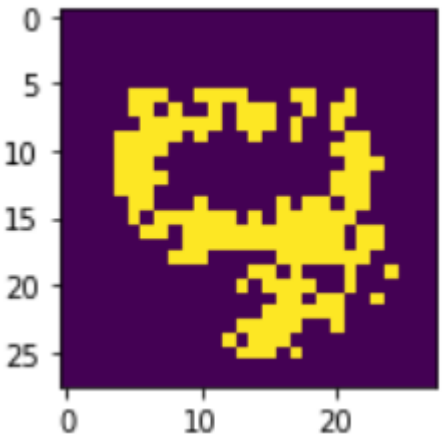
3.1 RBM for recognising MNIST images

Figure below shows the reconstruction errors for RBMs with hidden units 200, 300, 400, and 500 – using the contrastive divergence as training algorithm. We can see that the loss is continuously decreasing, thus converging. It is also fair to assume that the network is stable. In addition, the network performance is improved with the addition of units in the hidden layer.



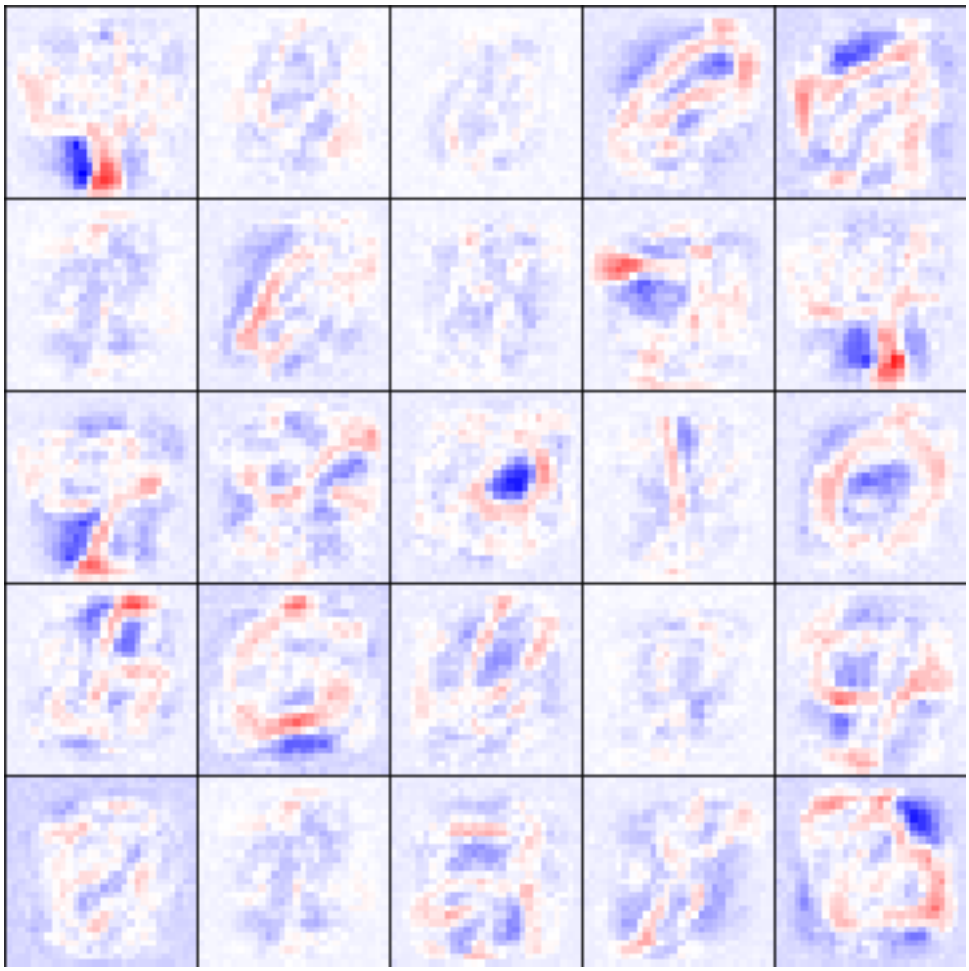
From analysing the weights, we also found that contrastive divergence has similar effect as regularization on the weights, as they either got pushed close to zero or increased in value after each epoch.

By looking further into the reconstruction capabilities of the network, we find that the network also has accurate recall of the digits – albeit with additional noise. Below we can see the digit 9 reconstructed by the RBM network.



When we look at the hidden activations of the hidden units for a batch of test data, we find that most of the hidden units activate on most images. In the image below, we can see each vertical line as the activations of units for one row of input, with white meaning activation and black meaning no activation. For some images, we can see that there were no activations. A possible explanation was that some units are specialized to detect certain features.

This explanation was confirmed when we visualized the receptive fields of the hidden units. As shown in the figure below, we can see that the units specialized in recognizing specific features of different numbers. For some, it looks like the units were specialized to detect a specific digit (such as 1 or 0), whereas with others it appears that the classification is deduced by combination of activations.



3.2 Towards deep networks - greedy layer-wise pretraining

Proceeding forward, we constructed DBN with two layers of RBMs and later with three RBMs. For a comparison between a DBN with two layers and just one RBM, we combined the reconstruction errors and compared them. The RBM had a total reconstruction loss of about 110k, whereas the DBN had a total reconstruction loss of about 140k. However, when comparing the reconstructed images, no discernible differences were found. Given the lower reconstruction error of the RBM, we can conclude that it is more efficient.

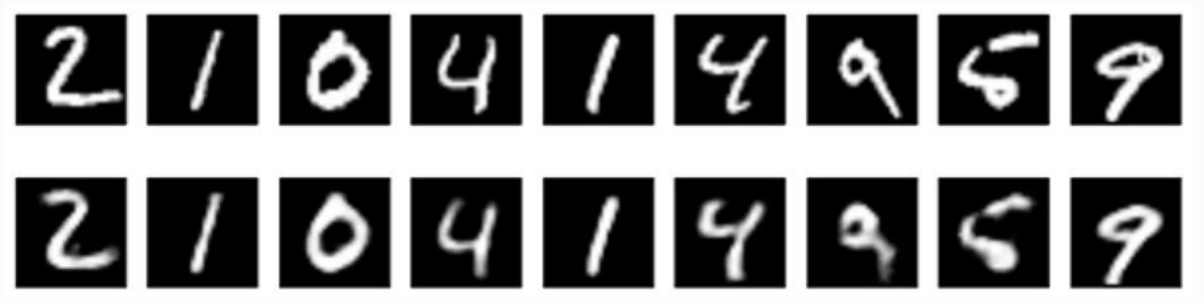
As proposed in the lab instructions, a DBN with three RBMs was constructed with the following architecture of 784 – 500 – 500 – 2000 hidden units. The final layer would receive labels and thus the architecture would be semi-supervised. Upon training with batch size 20 over 20 epochs, highest accuracy on the test data was about 73%. Using a batch size of 100, highest accuracy achieved was 85%.

Answering the question about what impacts the quality of our generated images; it was found that the number of iterations for Gibbs-Sampling, batch size, and epochs had the biggest impact in generating images with less noise. However, in most cases, the images looked too distorted to be reasonably from our training set.

3.3 Training with backpropagation

As we move from using a contrastive divergence training algorithm to a backpropagation training algorithm, we see several interesting results. Using the same MNIST dataset, we find that a simple autoencoder architecture with 32 units in the hidden layer, i.e: 784 – 32 – 784, greatly outperforms our previously built RBM and DBNs in reconstruction capabilities. We set the learning rate to 10e-5 and batch size to 256. Below we can see the reconstructions from our very simple autoencoder, with top row being our input data and bottom row being our reconstructed images.

From our autoencoder, we can already see significantly better reconstruction capabilities compared to our RBM and DBNs – however, let us see if we can improve our reconstruction by adding additional layers to our autoencoder. In this case, we use the architecture 128 – 64 – 32 – 64 – 128 – 784, with 784 as input. We also use Adadelta as optimization for our stochastic gradient descent. Below we can see our reconstructed images.



We cannot confidently conclude that our reconstruction got better, but it does seem to be the case. Looking at our test loss using TensorBoard, we can see a much smoother convergence of our model as well. While the training was not timed, it did seem significantly faster too – which further solidifies the autoencoders capabilities to reconstruct images in terms of efficiency with respect to time and loss.

Since the reconstructed images of the RBM and DBNs contained significant amount of noise whereas the autoencoder did not, we also wanted to demonstrate the latter's capabilities of removing nose. Again, we have our input in the upper row and our output in the bottom row.

4 Final remarks

To be honest, I do not have anything to add other than it was very interesting to play around with a Deep Belief Network – and to see how we've progressed since those times.